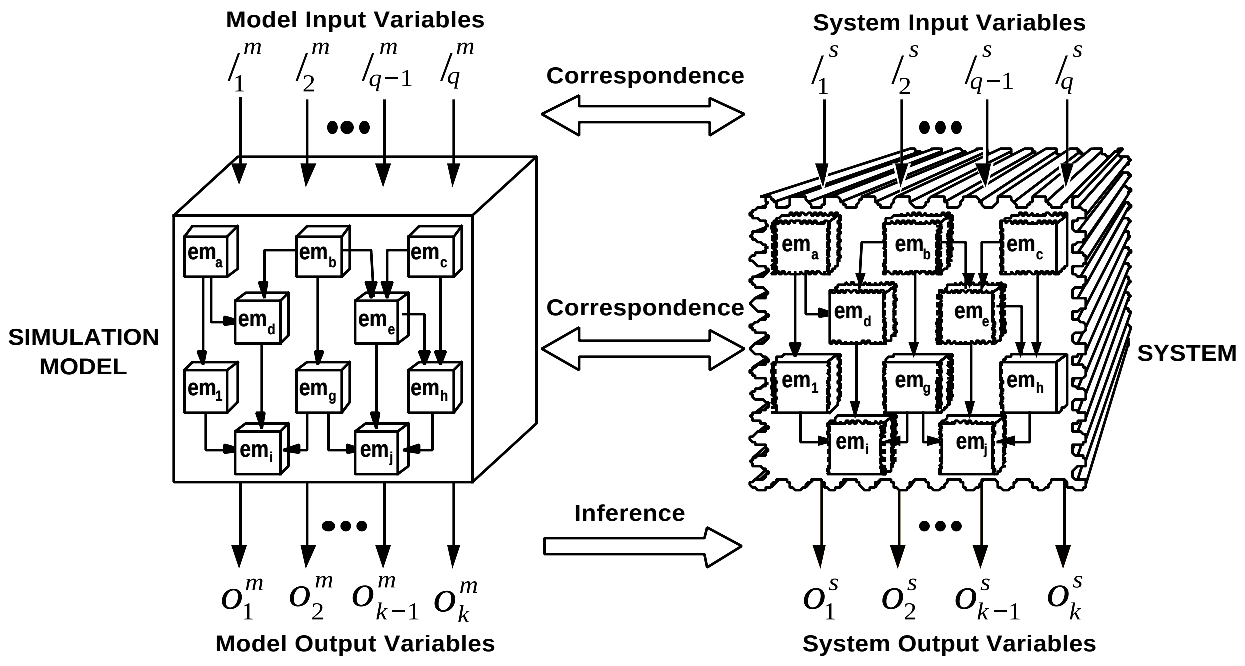
Model Input Variables
$I_1^m$ $I_2^m$ $I_{q-1}^m$ $I_q^m$
System Input Variables
$I_1^s$ $I_2^s$ $I_{q-1}^s$ $I_q^s$
Correspondence
Correspondence
Inference
SIMULATION MODEL
SYSTEM
$em_a$ $em_b$ $em_c$ $em_d$ $em_e$ $em_f$ $em_g$ $em_h$ $em_i$ $em_j$
$O_1^m$ $O_2^m$ $O_{k-1}^m$ $O_k^m$
Model Output Variables
$O_1^s$ $O_2^s$ $O_{k-1}^s$ $O_k^s$
System Output Variables

**Figure 4-3. Model and System Characteristics**

*In Approach 1, the m.r.a. is determined by the 100$(1-\gamma)$% s.c.i. for $\underline{\mu}^m - \underline{\mu}^s$ as*

$$\left[\underline{\delta} - \underline{\tau}\right] \qquad (1)$$

*where $\underline{\delta}' = \left[\delta_1, \delta_2, \ldots, \delta_k\right]$ represents lower bounds and $\underline{\tau}' = \left[\tau_1, \tau_2, \ldots, \tau_k\right]$ represents upper bounds of the s.c.i. The modeler can be 100$(1-\gamma)$% confident that the true differences between the population means of the model and system output variables are simultaneously contained within (1).*

*In Approach 2, the 100$(1-\gamma^m)$% s.c.i. are first constructed for $\underline{\mu}^m$ as*

$$\left[\underline{\delta}^m, \underline{\tau}^m\right] \qquad (2)$$

*where* $\left(\underline{\delta}^{m}\right)' = \left[\delta_1^m, \delta_2^m, \ldots, \delta_k^m\right]$ *and* $\left(\underline{\tau}^{m}\right)' = \left[\tau_1^m, \tau_2^m, \ldots, \tau_k^m\right]$. *Then, the* $100\left(1-\gamma^{s}\right)$*% s.c.i. are constructed for* $\underline{\mu}^{s}$ *as*

$$\left[\underline{\delta}^{s}, \underline{\tau}^{s}\right] \tag{3}$$

*where* $\left(\underline{\delta}^{s}\right)' = \left[\delta_1^s, \delta_2^s, \ldots, \delta_k^s\right]$ *and* $\left(\underline{\tau}^{s}\right)' = \left[\tau_1^s, \tau_2^s, \ldots, \tau_k^s\right]$. *Finally, using the Bonferroni inequality, the m.r.a. is determined by the following s.c.i. for* $\underline{\mu}^{m} - \underline{\mu}^{s}$ *with a confidence level of at least* $\left(1-\gamma^{m}-\gamma^{s}\right)$ *when the model and system outputs are dependent and with a level of at least* $\left(1-\gamma^{m}-\gamma^{s}+\gamma^{m}\gamma^{s}\right)$ *when the outputs are independent (Kleijnen, 1975):*

$$\left[\underline{\delta}^{m}-\underline{\tau}^{s}, \underline{\tau}^{m}-\underline{\delta}^{s}\right] \tag{4}$$

*In Approach 3, the model and system output variables are observed in pairs and the m.r.a. is determined by the 100*$\left(1-\gamma\right)$*% s.c.i. for* $\underline{\mu}^{d}$*, the population means of the differences of paired observations, as*

$$\left[\underline{\delta}^{d}, \underline{\tau}^{d}\right] \tag{5}$$

*where* $\left(\underline{\delta}^{d}\right)' = \left[\delta_1^d, \delta_2^d, \ldots, \delta_k^d\right]$ *and* $\left(\underline{\tau}^{d}\right)' = \left[\tau_1^d, \tau_2^d, \ldots, \tau_k^d\right]$.

*The m.r.a. is constructed with the observations derived from the model and system output variables by running the model with the same input data and operational conditions that drive the real system. If the simulation is self-driven, then the model input data come independently from the same populations or stochastic process as the system input data. Because the model and system input data are independent of each other, but come from the same populations, the model and system output data are expected to be independent and identically distributed. Hence, Approach 1 or 2 can be used. The use of Approach 3 in this case would be less efficient. If the simulation is trace-driven, the model input data are exactly the same as the system input data. In this case, the model and system output data are* *expected to be dependent and identical. Therefore, Approach 2 or 3 should be used.*

*Sometimes, the model or simulation application sponsor or proponent may specify an acceptable range of accuracy for a specific simulation. This specification can be made for the mean behavior of a stochastic simulation as*

$$\underline{L} \leq \underline{\mu}^{m} - \underline{\mu}^{s} \leq \underline{U}. \qquad (6)$$

*where* $\underline{L}' = [L_1, L_2, \ldots, L_k]$ *and* $\underline{U}' = [U_1, U_2, \ldots, U_k]$ *are the lower and upper bounds of the acceptable differences between the population means of the model and system output variables. In this case, the m.r.a. should be compared against Equation (6) to evaluate model validity.*

*The shorter the lengths of the m.r.a., the more meaningful is the information they provide. The lengths can be decreased by increasing the sample sizes or by decreasing the confidence level. Such increases in sample sizes, however, may increase the cost of data collection. Thus, a trade-off analysis may be necessary among the sample sizes, confidence levels, half-length estimates of the m.r.a., data collection method, and cost of data collection. For details of performing the trade-off analysis, see Balci and Sargent, 1984.*

### 4.1.3.23 Structural Testing

Structural testing (also called *white-box testing*) consists of six testing techniques: branch, condition, data flow, loop, path, and statement testing. Structural (white-box) testing evaluates the model based on its internal structure (how it is built), whereas functional (black-box) testing assesses the input-output transformation accuracy of the model. (Refer to Section 4.1.3.12.) Structural testing employs data flow and control flow diagrams to assess the accuracy of internal model structure by examining model elements such as statements, branches, conditions, loops, internal logic, internal data representations, submodel interfaces, and model execution paths.

**Branch testing** runs the model or simulation under test data to execute as many branch alternatives as possible, as many times as possible, and to substantiate their accurate operation. The more branches that test successfully, the more confidence is gained in the model's accurate execution with respect to its logical branches (Beizer, 1990).

**Condition testing** runs the model or simulation under test data to execute as many logical conditions as possible, as many times as possible, and to substantiate their accurate operation. The more logical conditions that test successfully, the more confidence is gained in the model's accurate execution with respect to its logical conditions.

**Data flow testing** uses the control flowgraph to explore sequences of events related to the status of data structures and to examine data-flow anomalies. For example, sufficient paths can be forced to execute under test data to ensure that every data element and structure is initialized before use or every declared data structure is used at least once in an executed path (Beizer, 1990).

**Loop testing** runs the model or simulation under test data to execute as many loop structures as possible, as many times as possible, and to substantiate their accurate operation. The more loop structures that test successfully, the more confidence is gained in the model's accurate execution with respect to its loop structures (Pressman, 1996).

**Path testing** runs the model or simulation under test data to execute as many control flow paths as possible, as many times as possible, and to substantiate their accurate operation. The more control flow paths that test successfully, the more confidence is gained in the model's accurate execution with respect to its control flow paths, but 100 percent path coverage is impossible to achieve for a reasonably large M&S application (Beizer, 1990).

Path testing is performed in three steps (Howden, 1976). In Step 1, the model control structure is determined and represented in a control flow diagram. In Step 2, test data is generated to cause selected model logical paths to be executed. Symbolic evaluation (Section 4.1.2.8) can be used to identify and classify input data based on the symbolic representation of the model. The test data is generated in such a way as to (a) cover all statements in the path, (b) encounter all nodes in the path, (c) cover all branches from a node in the path, (d) achieve all decision combinations at each branch point in the path, and (e) traverse all paths (Prather and Myers, 1987). In Step 3, by using the generated test data, the model is forced to proceed through each path in its execution structure, thereby providing comprehensive testing.

In practice, only a subset of all possible model paths is selected for testing due to budgetary constraints. Recent work has sought to increase the amount of coverage per test case or to improve the effectiveness of the testing by selecting the most critical areas to test. (Savvy readers may note that this technique is similar to the larger concept of VV&A tailoring that was addressed in Chapters 1 and 3.) The path prefix strategy is an adaptive strategy that uses previously tested paths as a guide in the selection of subsequent test paths. Prather and Myers (1987) prove that the path prefix strategy achieves total branch coverage.

The identification of essential paths is a strategy that reduces the path coverage required by nearly 40 percent (Chusho, 1987) by eliminating nonessential paths. Paths overlapped by other paths are nonessential. The model control flow graph is transformed into a directed graph whose arcs (called *primitive arcs*) correspond to the essential paths of the model. Nonessential arcs are called *inheritor arcs* because they inherit information from the primitive arcs. The graph produced during the transformation is called an *inheritor-reduced graph.* Chusho (1987) presents algorithms for efficiently identifying nonessential paths, reducing the control graph into an inheritor-reduced graph, and applying the concept of essential paths to the selection of effective test data.

**Statement testing** runs the model or simulation under test data to execute as many statements as possible, as many times as possible, and to substantiate their accurate operation. The more statements that test successfully, the more confidence is gained in the model's accurate execution with respect to its statements (Beizer, 1990).

#### *4.1.3.24 Submodel/Module Testing*

Submodel testing requires a top-down decomposition of the model into submodels. The executable model is instrumented to collect data on all input and output variables of a submodel. The system is instrumented (if possible) to collect similar data. Then, the behavior of each submodel is compared with the corresponding subsystem's behavior to judge the submodel's validity. If a subsystem can be modeled analytically, its exact solution can be compared against the simulation solution to assess its validity quantitatively.

As enumerated in Principle 6 in Chapter 2, validating each submodel individually does not imply sufficient validity for the whole model. Each submodel is found sufficiently valid with some allowable error. The allowable errors can accumulate to make the whole model invalid. Therefore, after each submodel is validated, the whole model itself must be tested.

### 4.1.3.25 Symbolic Debugging

This technique employs a debugging tool that allows the modeler to manipulate model execution while viewing the model at the source code level. By setting *breakpoints*, the modeler can interact with the entire model one step at a time, at predetermined locations, or under specified conditions. While using a symbolic debugger, the modeler may alter model data values or replay a portion of the model, i.e., execute it again under the same conditions. Typically, the modeler utilizes the information gathered with execution testing techniques (see Section 4.1.3.9) to isolate a problem or its proximity. Then the debugger is employed to determine how and why the error occurs.

Current state-of-the-art debuggers can view the runtime code as it appears in the source listing, set *watch* variables to monitor data flow, examine complex data structures, and even communicate with asynchronous input/output channels. The use of symbolic debugging can reduce greatly the debugging effort while increasing its effectiveness. Symbolic debugging allows the modeler to locate errors and check numerous circumstances that lead to errors (Whitner and Balci, 1989).

### 4.1.3.26 Top-Down Testing

Top-down testing is used with top-down model development. In top-down development, model construction starts with the submodels at the highest level and culminates with the routines at the base level, i.e., the ones that cannot be decomposed further. As each submodel is completed, it is tested thoroughly. When submodels with the same parent have been developed and tested, the submodels are integrated and their integration is tested. This process is repeated until the whole model has been integrated and tested. The integration of completed submodels need not wait for all submodels at the same level to be completed. Submodel integration and testing can be, and often is, performed incrementally (Sommerville, 1996).

Top-down testing begins with a test of the global model at its highest level. When testing a given level, calls to submodels at lower levels are simulated using *stubs*. A stub is a dummy submodel that has no function other than to let its caller complete the call. Fairley (1976) lists the following advantages of top-down testing: (a) model integration testing is minimized; (b) a working model is produced earlier in the development process; (c) higher level interfaces are tested first; (d) a natural environment for testing lower levels is created; and (e) errors are localized to new submodels and interfaces.

Some of the disadvantages of top-down testing are (a) thorough submodel testing is discouraged, because the entire model must be executed to perform testing; (b) testing can be expensive, because the whole model must be executed for each test; (c) adequate input data is difficult to obtain because of the complexity of the data paths and control predicates; and (d) integration testing is hampered because of the size and complexity of testing the whole model (Fairley, 1976).

#### *4.1.3.27 Visualization/Animation*

Visualization and animation of a simulation greatly assist in model V&V (Sargent, 1992). Displaying graphical images of internal (e.g., how customers are served by a cashier) and external (e.g., utilization of the cashier) dynamic behavior of a model during execution exhibits errors. For example, in visual simulation of a traffic intersection, the modeler can observe the arrival of vehicles in different lanes and their movements through the intersection as the traffic light changes. Visualizing the model as it executes and comparing it with the real traffic intersection can help identify discrepancies between the model and the system.

Seeing the model in action is very useful for uncovering errors; however, it does not guarantee model correctness (Paul, 1989). Therefore, visualization should be used with caution.

### 4.1.4 Formal V&V Techniques

Formal V&V techniques are based on formal mathematical proofs of correctness. If attainable, a formal proof of correctness is the most effective means of model V&V. Unfortunately, *if attainable* is the sticking point. Current formal proof of correctness techniques cannot be applied to even a reasonably complex M&S application; however, formal techniques serve as the foundation for other V&V techniques. The most commonly known eight techniques are briefly described below: (a) induction, (b) inference, (b) logical deduction, (d) inductive assertions, (e) lambda-calculus, (f) predicate calculus, (g) predicate transformation, and (h) proof of correctness (Khanna, 1991; Whitner and Balci, 1989).

**Induction, inference**, and **logical deduction** are simply acts of justifying conclusions on the basis of premises given. An argument is valid if the steps used to progress from the premises to the conclusion conform to established *rules of inference*. Inductive reasoning is based on invariant properties of a set of observations; assertions are invariants because their value is defined to be true. Given that the initial model assertion is correct, it stands to reason that if each path progressing from that assertion is correct and each path subsequently progressing from the previous assertion is correct, then the model must be correct if it terminates. Birta and Ozmizrak (1996) present a knowledge-based approach for M&S validation that uses a validation knowledge base containing rules of inference.

**Inductive assertions** assess model correctness based on an approach that is very close to formal proof of model correctness. It is conducted in three steps. In Step 1, input-to-output relations for all model variables are identified. In Step 2, these relations are converted into assertion statements and are placed along the model execution paths so that an assertion statement lies at the beginning and end of each model execution path. In Step 3, verification is achieved by proving for each path that, if the assertion at the beginning of the path is true and all statements along the path are executed, then the assertion at the end of the path is true. If all paths plus model termination can be proved, by induction, the model is proved to be correct (Manna *et al.,* 1973; Reynolds and Yeh, 1976).

**Lambda Calculus** (Barendregt, 1981) is a system that transforms the model into formal expressions by rewriting strings. The model itself can be considered a large string. Lambda calculus specifies rules for rewriting strings to transform the model into lambda calculus expressions. Using lambda calculus, the modeler can express the model formally to apply mathematical proof of correctness techniques to it.

**Predicate calculus** provides rules for manipulating predicates. A predicate is a combination of simple relations, such as *completed_jobs >steady_state_length*. A predicate will be either true or false. The model can be defined in terms of predicates and manipulated using the rules of predicate calculus. Predicate calculus forms the basis of all formal specification languages (Backhouse, 1986).

**Predicate transformation** (Dijkstra, 1975; Yeh, 1977) verifies model correctness by formally defining the semantics of the model with a mapping that transforms model output states to all possible model input states. This representation is the basis from which model correctness is proved.

Formal **proof of correctness** expresses the model in a precise notation and then mathematically proves that the executed model terminates and satisfies the requirements with sufficient accuracy (Backhouse, 1986; Schach, 1996). Attaining proof of correctness in a realistic sense is not possible under the current state of the art. The advantage of realizing proof of correctness is so great, however, that, when the capability is realized, it will revolutionize V&V.

## 4.2 Guidelines for Using the V&V Techniques

It is very important to understand the twelve principles of VV&A presented in Chapter 2 when applying the techniques just described to the VV&A process presented in Chapter 3. The principles help researchers, practitioners, and managers better understand M&S VV&A. They provide the underpinnings for the V&V techniques. Understanding and applying the principles is crucially important for the success of an M&S application.

Recall that, as stated in Principle 2 of Chapter 2, V&V is not a phase or step in the M&S life cycle but a continuous activity throughout the entire M&S life cycle. Table 4-2 shows the techniques that apply to the major stages of the generic VV&A process:

- Problem Definition
- M&S Approach
- M&S Requirements
- Conceptual Model
- M&S Design
- M&S Implementation
- M&S Application
- M&S Acceptability Assessment

The rows of Table 4-2 list the 76 V&V techniques described in this chapter, including a placeholder for the 18 statistical techniques shown in Table 4-1. These statistical techniques can be used to perform model validation quantitatively if data can be collected on the input and output processes of the system.

**Table 4-2. Applicability of the V&V Techniques Throughout the M&S Life Cycle**

| | *Formul. Problem* | *M&S Approach* | *M&S Reqs.* | *Concep. Model* | *M&S Design* | *M&S Implem.* | *M&S Applic.* | *M&S Accept. Assess.* |
|---|---|---|---|---|---|---|---|---|
| Acceptance Testing | | | | | | | ◆ | ◆ |
| Alpha Testing | | | | | | ◆ | ◆ | |
| Assertion Checking | | | | | | ◆ | | |
| Audit | ◆ | ◆ | ◆ | ◆ | ◆ | ◆ | | |
| Authorization Testing | | | | | | ◆ | ◆ | ◆ |
| Beta Testing | | | | | | ◆ | ◆ | |
| Bottom-Up Testing | | | | | | ◆ | | |
| Boundary Value Testing | | | | | | ◆ | | |
| Branch Testing | | | | | | ◆ | | |
| Calling Structure Analysis | | | | ◆ | ◆ | ◆ | | |
| Cause-Effect Graphing | ◆ | | ◆ | ◆ | ◆ | ◆ | | |
| Comparison Testing | | ◆ | ◆ | ◆ | ◆ | ◆ | ◆ | |
| Concurrent Process Analysis | | | | | | ◆ | ◆ | |
| Condition Testing | | | | | | ◆ | | |
| Control Flow Analysis | | | | ◆ | ◆ | ◆ | | |
| Data Dependency Analysis | ◆ | | | ◆ | ◆ | ◆ | | |
| Data Flow Analysis | | | | ◆ | ◆ | ◆ | | |
| Data Flow Testing | | | | | | ◆ | | |
| Data Interface Testing | | | | | | ◆ | ◆ | |
| Debugging | | | | | | ◆ | | |
| Desk Checking | ◆ | ◆ | ◆ | ◆ | ◆ | ◆ | | |
| Equivalence Partitioning Testing | | | | | | ◆ | | |
| Execution Monitoring | | | | | | ◆ | ◆ | |
| Execution Profiling | | | | | | ◆ | ◆ | |
| Execution Tracing | | | | | | ◆ | ◆ | |
| Extreme Input Testing | | | | | | ◆ | | |
| Face Validation | ◆ | ◆ | ◆ | ◆ | ◆ | ◆ | ◆ | ◆ |

**Table 4-2. Applicability of the V&V Techniques Throughout the M&S Life Cycle (cont.)**

| | *Formul. Problem* | *M&S Approach* | *M&S Reqs.* | *Concep. Model* | *M&S Design* | *M&S Implem.* | *M&S Applic.* | *M&S Accept. Assess.* |
|---|---|---|---|---|---|---|---|---|
| Fault/Failure Analysis | | | | | | ◆ | ◆ | |
| Fault/Failure Insertion Testing | | | | | | ◆ | ◆ | |
| Field Testing | | | | | | | ◆ | |
| Functional Testing | | | | | | ◆ | ◆ | |
| Graphical Comparisons | | | | | | ◆ | ◆ | |
| Induction | | | | ◆ | ◆ | | | |
| Inductive Assertions | | | | ◆ | ◆ | | | |
| Inference | | | | ◆ | ◆ | | | |
| Inspections | ◆ | ◆ | ◆ | ◆ | ◆ | ◆ | ◆ | ◆ |
| Invalid Input Testing | | | | | | ◆ | ◆ | |
| Lambda Calculus | | | | ◆ | ◆ | | | |
| Logical Deduction | | | | ◆ | ◆ | | | |
| Loop Testing | | | | | | ◆ | | |
| Model Interface Analysis | | | ◆ | ◆ | ◆ | | | |
| Model Interface Testing | | | | | | ◆ | ◆ | ◆ |
| Object-Flow Testing | | | | | | ◆ | ◆ | ◆ |
| Partition Testing | | | | | ◆ | ◆ | | |
| Path Testing | | | | | | ◆ | ◆ | |
| Performance Testing | | | | | | | ◆ | ◆ |
| Predicate Calculus | | | | ◆ | ◆ | | | |
| Predicate Transformation | | | | ◆ | ◆ | | | |
| Predictive Validation | | | | | | ◆ | ◆ | ◆ |
| Product Testing | | | | | | | ◆ | ◆ |
| Proof of Correctness | | | | ◆ | ◆ | | | |
| Real-Time Input Testing | | | | | | ◆ | ◆ | ◆ |
| Regression Testing | | | | | | ◆ | ◆ | |
| Reviews | ◆ | ◆ | ◆ | ◆ | ◆ | ◆ | ◆ | ◆ |
| Security Testing | | | | | | | ◆ | ◆ |

**Table 4-2. Applicability of the V&V Techniques Throughout the M&S Life Cycle (cont.)**

| | *Formul. Problem* | *M&S Approac h* | *M&S Reqs.* | *Concep. Model* | *M&S Design* | *M&S Implem.* | *M&S Applic.* | *M&S Accept. Assess.* |
|---|---|---|---|---|---|---|---|---|
| Self-Driven Input Testing | | | | | | ♦ | ♦ | ♦ |
| Semantic Analysis | | | | | ♦ | ♦ | | |
| Sensitivity Analysis | | | | | | ♦ | ♦ | ♦ |
| Standards Testing | | | | | | | ♦ | ♦ |
| State Transition Analysis | | | | ♦ | ♦ | ♦ | | |
| Statement Testing | | | | | | ♦ | | |
| Statistical Techniques (Table 4-1) | | | | | | ♦ | ♦ | ♦ |
| Stress Testing | | | | | | ♦ | ♦ | ♦ |
| Structural Analysis | | | | ♦ | ♦ | | | |
| Submodel/Module Testing | | | | | | ♦ | | |
| Symbolic Debugging | | | | | | ♦ | | |
| Symbolic Evaluation | | | | | ♦ | | | |
| Syntax Analysis | | | | | | ♦ | | |
| Top-Down Testing | | | | | | ♦ | | |
| Trace-Driven Input Testing | | | | | | ♦ | ♦ | ♦ |
| Traceability Assessment | | | | ♦ | ♦ | ♦ | ♦ | |
| Turing Test | | | | | | ♦ | ♦ | ♦ |
| User Interface Analysis | | | | | | ♦ | ♦ | ♦ |
| User Interface Testing | | | | | | ♦ | ♦ | ♦ |
| Visualization/Animation | | | | | | ♦ | ♦ | ♦ |
| Walkthroughs | ♦ | ♦ | ♦ | ♦ | ♦ | ♦ | ♦ | ♦ |

Table 4-2 can be used to determine the V&V techniques that apply to each major stage of the M&S life cycle. From those applicable, the technique(s) for a particular V&V activity can be selected by considering the following: the model type as described in Figure 3-1; the problem to be solved through the use of M&S; the specific objectives of the M&S application; and the constraints of the application, including time, cost, and schedule.

The life cycle application of V&V is extremely important for successful completion of complex and large-scale M&S applications. How much to test or when to stop testing depends on the M&S application objectives. The V&V effort should continue until the modeler obtains sufficient confidence in the credibility and acceptability of the model or simulation results. *Sufficient confidence* is determined by the objectives of the M&S application.

Yet, it is recognized that applying V&V techniques throughout the life cycle is time-consuming and can be costly if not properly tailored to the relevant requirements of the problem. In practice, under pressure to complete an M&S application within a given timeframe, VV&A is usually sacrificed first. The sacrifice of VV&A means less than the delivery of a model without proven credibility and therefore without value to the decision maker. Remember the bottom line from Principle 2: Correction of errors early in development always costs less than correction of errors later. If you are worried about the cost of VV&A, it is better to spend a little up front than a lot later. During a meeting in the General's office or standing before a senior-level review board is not the time to realize that the sacrifice of VV&A was a mistake.

## References

Ackerman, A.F., Fowler, P.J., & Ebenau, R.G. (1983). Software inspections and the industrial production of software. In Hans-Ludwig Hausen (Ed.), *Software validation: Inspection, testing, verification, alternatives.* Proceedings of the Symposium on Software Validation (pp. 13–40). Darmstadt, FRG.

Adrion, W.R., Branstad, M.A., & Cherniavsky, J.C. (1982). Validation, verification, and testing of computer software. *Computing Surveys, 14* (2), 159–192.

Aigner, D.J. (1972). A note on verification of computer simulation models. *Management Science, 18* (11), 615–619.

Allen, F.E. & Cocke, J. (1976). A program data flow analysis procedure. *Communications of the ACM, 19* (3), 137–147.

Backhouse, R.C. (1986). *Program construction and verification.* London: Prentice-Hall International (UK) Ltd.

Balci, O. (1988). The implementation of four conceptual frameworks for simulation modeling in high-level languages. In M.A. Abrams, P.L. Haigh, & J.C. Comfort (Eds.), *Proceedings of the 1988 Winter Simulation Conference* (pp. 287–295). Piscataway, NJ: IEEE.

Balci, O., Bertelrud, A.I., Esterbrook, C.M., & Nance, R.E. (1995). A picture-based object-oriented visual simulation environment. In C. Alexopoulos, K. Kang, W.R. Lilegdon, & D. Goldsman (Eds.), *Proceedings of the 1995 Winter Simulation Conference* (pp. 1333–1340). Piscataway, NJ: IEEE.

Balci, O. & Sargent, R.G. (1981). A methodology for cost-risk analysis in the statistical validation of simulation models. *Communications of the ACM, 24* (4), 190–197.

Balci, O. & Sargent, R.G. (1982a). Some examples of simulation model validation using hypothesis testing. In H.J. Highland, Y.W. Chao, & O.S. Madrigal (Eds.), *Proceedings of the 1982 Winter Simulation Conference* (pp. 620–620). Piscataway, NJ: North-Holland IEEE.

Balci, O. & Sargent, R.G. (1982b). Validation of multivariate response models using Hotelling's two-sample $T^2$ test. *Simulation, 39* (6), 185–192.

Balci, O. & Sargent, R.G. (1983). Validation of multivariate response trace-driven simulation models. In A.K. Agrawala & S.K. Tripathi (Eds.), *Performance '83* (pp. 309–323). North-Holland, Amsterdam.

Balci, O. & Sargent, R.G. (1984). Validation of simulation models via simultaneous confidence intervals. *American Journal of Mathematical and Management Sciences, 4* (3&4), 375–406.

Banks, J., Carson, J.S., & Nelson, B.L. (1996). *Discrete-event system simulation* (2nd ed.). Englewood Cliffs, NJ: Prentice-Hall.

Barendregt, H.P. (1981). *The lambda calculus: Its syntax and semantics.* New York: North-Holland.

Beizer, B. (1990). *Software testing techniques* (2$^{nd}$ ed.). New York: Van Nostrand Reinhold.

Birta, L.G. & Ozmizrak, F.N. (1996). A knowledge-based approach for the validation of simulation models: The foundation. *ACM Transactions on Modeling and Computer Simulation* (in press).

Chusho, T. (1987). Test data selection and quality estimation based on the concept of essential branches for path testing. *IEEE Transactions on Software Engineering, SE-13* (5), 509–517.

Cohen, K.J. & Cyert, R.M. (1961). Computer models in dynamic economics. *Quarterly Journal of Economics, 75* (1), 112–127.

Damborg, M.J. & Fuller, L.F. (1976). Model validation using time and frequency domain error measures (ERDA Report No. 76-152). Springfield, VA: NTIS.

Deutsch, M.S. (1982). *Software verification and validation: Realistic project approaches.* Englewood Cliffs, NJ: Prentice-Hall.

Dillon, L.K. (1990). Using symbolic execution for verification of Ada tasking programs. *ACM Transactions on Programming Languages and Systems, 12* (4), 643–669.

Dobbins, J.H. (1987). Inspections as an up-front quality technique. In G.G. Schulmeyer & J.I. McManus (Eds.), *Handbook of software quality assurance* (pp. 137–177). New York: Van Nostrand-Reinhold Company.

Dunn, R.H. (1984). *Software defect removal*, New York: McGraw-Hill.

Dunn, R.H. (1987). The quest for software reliability. In G.G. Schulmeyer & J.I. McManus (Eds.), *Handbook of software quality assurance* (pp. 342–384). New York: Van Nostrand-Reinhold Company.

Emshoff, J.R. & Sisson, R.L. (1970). *Design and use of computer simulation models.* New York: MacMillan.

Fairley, R.E. (1975). An experimental program-testing facility. *IEEE Transactions on Software Engineering, SE-1* (4), 350–357.

Fairley, R.E. (1976, July). Dynamic testing of simulation software. In *Proceedings of the 1976 Summer Computer Simulation Conference* (pp. 708–710). La Jolla, CA: Simulation Councils.

Fishman, G.S. &. Kiviat, P.J. (1967). The analysis of simulation generated time series. *Management Science, 13* (7), 525–557.

Forrester, J.W. (1961). *Industrial dynamics.* Cambridge, MA: MIT Press.

Fujimoto, R.M. (1990). Parallel discrete event simulation. *Communications of the ACM, 33* (10), 31–53.

Fujimoto, R.M. (1993). Parallel discrete event simulation: Will the field survive? *ORSA Journal on Computing, 5* (3), 213–230.

Gafarian, A.V. & Walsh, J.E. (1969). Statistical approach for validating simulation models by comparison with operational systems. In *Proceedings of the 4th International Conference on Operations Research* (pp. 702–705). New York: John Wiley & Sons.

Gallant, A.R., Gerig, T.M., & Evans, J.W. (1974). Time series realizations obtained according to an experimental design. *Journal of the American Statistical Association, 69* (347), 639–645.

Garratt, M. (1974, July). Statistical validation of simulation models. In *Proceedings of the 1974 Summer Computer Simulation Conference* (pp. 915–926). La Jolla, CA: Simulation Councils.

Hermann, C.F. (1967). Validation problems in games and simulations with special reference to models of international politics. *Behavioral Science, 12* (3), 216–231.

Hollocker, C.P. (1987). The standardization of software reviews and audits. In G.G. Schulmeyer & J.I. McManus (Eds.), *Handbook of software quality assurance* (pp. 211–266). New York: Van Nostrand-Reinhold Company.

Howden, W.E. (1976). Reliability of the path analysis testing strategy. *IEEE Transactions on Software Engineering, SE-2* (3), 208–214.

Howden, W.E. (1980). Functional program testing. *IEEE Transactions on Software Engineering, SE-6* (2), 162–169.

Howrey, P. & Kelejian, H.H. (1969). Simulation versus analytical solutions. In T.H. Naylor (Ed.), *The design of computer simulation experiments* (pp. 207–231). Durham, NC: Duke University Press.

Hunt, A.W. (1970). Statistical evaluation and verification of digital simulation models through spectral analysis. Unpublished doctoral dissertation, University of Texas at Austin.

Khanna, S. (1991). Logic programming for software verification and testing. *The Computer Journal, 34* (4), 350–357.

Kheir, N.A. & Holmes, W.M. (1978). On validating simulation models of missile systems. *Simulation, 30* (4), 117–128.

King, J.C. (1976). Symbolic execution and program testing. *Communications of the ACM, 19* (7), 385–394.

Kleijnen, J.P.C. (1975). *Statistical techniques in simulation* (Vol. 2). New York: Marcel Dekker.

Knight, J.C. & Myers, E.A. (1993). An improved inspection technique. *Communications of the ACM, 36* (11), 51–61.

Law, A.M. & Kelton, W.D. (1991). *Simulation modeling and analysis* (2$^{nd}$ ed.). New York: McGraw-Hill.

Manna, Z., Ness, S., & Vuillemin, J. (1973). Inductive methods for proving properties of programs. *Communications of the ACM, 16* (8), 491–502.

Miller, D.K. (1975). Validation of computer simulations in the social sciences. In *Proceedings of the Sixth Annual Conference on Modeling and Simulation* (pp. 743–746). Pittsburg, PA.

Miller, D.R. (1974a, July). Model validation through sensitivity analysis. In *Proceedings of the 1974 Summer Computer Simulation Conference* (pp. 911–914). La Jolla, CA: Simulation Councils.

Miller, D.R. (1974b). Sensitivity analysis and validation of simulation models. *Journal of Theoretical Biology, 48* (2), 345–360.

Miller, L.A., Groundwater, E.H., Hayes, J.E., & Mirsky, S.M. (1995). Survey and assessment of conventional software verification and validation methods (Special Publication NUREG/CR-6316, Vol. 2). Washington, DC: U.S. Nuclear Regulatory Commission.

Myers, G.J. (1978). A controlled experiment in program testing and code walkthroughs/inspections. *Communications of the ACM, 21* (9), 760–768.

Myers, G.J. (1979). *The art of software testing.* New York: John Wiley & Sons.

Naylor, T.H. & Finger, J.M. (1967). Verification of computer simulation models. *Management Science, 14* (2), B92–B101.

Ould, M.A. & Unwin, C. (1986). *Testing in software development.* Great Britain: Cambridge University Press.

Page, E.H. & Nance, R.E. (1994, July). Parallel discrete event simulation: A modeling methodological perspective. In D.K. Arvind, R. Bagrodia, & J.Y-B. Lin (Eds.), *Proceedings of the Eighth Workshop in Parallel and Distributed Simulation* (PADS ‘94) (pp. 88–93). Los Alamitos, CA: IEEE Computer Society Press.

Paul, R.J. (1989). Visual simulation: Seeing is believing? In R. Sharda, B.L. Golden, E. Wasil, O. Balci, & W. Stewart (Eds.), *Impacts of recent computer advances on operations research* (pp. 422–432). New York: Elsevier.

Perry, W. (1995). *Effective methods for software testing.* New York: John Wiley & Sons.

Prather, R.E. & Myers, J.P., Jr. (1987). The path prefix software testing strategy. *IEEE Transactions on Software Engineering, SE-13* (7), 761–766.

Pressman, R.S. (1996). *Software engineering: A practitioner’s approach* (4th Ed.). New York: McGraw-Hill.

Ramamoorthy, C.V., Ho, S.F., & Chen, W.T. (1976). On the automated generation of program test data. *IEEE Transactions on Software Engineering, SE-2* (4), 293–300.

Rattray, C. (Ed.). (1990). *Specification and verification of concurrent systems.* New York: Springer-Verlag.

Reynolds, C. & Yeh, R.T. (1976). Induction as the basis for program verification. *IEEE Transactions on Software Engineering, SE-2* (4), 244–252.

Richardson, D.J. & Clarke, L.A. (1985). Partition analysis: A method combining testing and verification. *IEEE Transactions on Software Engineering, SE-11* (12), 1477–1490.

Rowland, J.R. & Holmes, W.M. (1978) Simulation validation with sparse random data. *Computers and Electrical Engineering, 5* (3), 37–49.

Sargent, R.G. (1992). Validation and verification of simulation models. In J.J. Swain, D. Goldsman, R.C. Crain, & J.R. Wilson (Eds.), *Proceedings of the 1992 Winter Simulation Conference* (pp. 104–114). Piscataway, NJ: IEEE.

Schach, S.R. (1996). *Software engineering* (3rd ed.). Homewood, IL: Irwin.

Schruben, L.W. (1980). Establishing the credibility of simulations. *Simulation, 34* (3), 101–105.

Shannon, R.E. (1975). *Systems simulation: The art and science.* Englewood Cliffs, NJ: Prentice-Hall.

Sommerville, I. (1996). *Software engineering* (5th ed.). Reading, MA: Addison-Wesley.

Teorey, T.J. (1975). Validation criteria for computer system simulations. *Simuletter, 6* (4), 9–20.

Theil, H. (1961). *Economic forecasts and policy.* Amsterdam, The Netherlands: North-Holland.

Turing, A.M. (1963). Computing machinery and intelligence. In E.A. Feigenbaum & J. Feldman (Eds.), *Computers and thought* (pp. 11–15). New York: McGraw-Hill.

Tytula, T.P. (1978, June). A method for validating missile system simulation models (Technical Report E-78-11). Redstone Arsenal, AL: U.S. Army Missile R&D Command.

Van Horn, R.L. (1971). Validation of simulation results. *Management Science, 17* (5), 247–258.

Watts, D. (1969). Time series analysis. In T.H. Taylor (Ed.), *The design of computer simulation experiments* (pp. 165–179). Durham, NC: Duke University Press.

Whitner, R.B. & Balci, O. (1989). Guidelines for selecting and using simulation model verification techniques. In E.A. MacNair, K.J. Musselman, & P. Heidelberger (Eds.), *Proceedings of the 1989 Winter Simulation Conference* (pp. 559–568). Piscataway, NJ: IEEE.

Wright, R.D. (1972). Validating dynamic models: An evaluation of tests of predictive power. In *Proceedings of the 1972 Summer Computer Simulation Conference* (pp. 1286–1296). La Jolla, CA: Simulation Councils.

Yourdon, E. (1985). *Structured walkthroughs* (3rd ed.). New York: Yourdon Press.

Yucesan, E. & Jacobson, S.H. (1992). Building correct simulation models is difficult. In J.J. Swain, D. Goldsman, R.C. Crain, & J.R. Wilson (Eds.), *Proceedings of the 1992 Winter Simulation Conference* (pp. 783–790). Piscataway, NJ: IEEE.

Yucesan, E. & Jacobson, S.H. (1996). Intractable structural issues in discrete event simulation: Special cases and heuristic approaches. *ACM Transactions on Modeling and Computer Simulation* (in press).

## Additional Reading

Balci, O. (1986). Requirements for model development environments. *Computers & Operations Research, 13* (1), 53–67.

Balci, O. & Nance, R.E. (1987). Simulation model development environments: A research prototype. *Journal of Operational Research Society, 38* (8), 753–763.

Derrick, E.J. & Balci, O. (1995). A visual simulation support environment based on the DOMINO conceptual framework. *Journal of Systems and Software, 31* (3), 215–237.

Dijkstra, E.W. (1975). Guarded commands, non-determinacy and a calculus for the derivation of programs. *Communications of the ACM, 18* (8), 453–457.

Moose, R.L. & Nance, R.E. (1989). The design and development of an analyzer for discrete event model specifications. In R. Sharda, B.L. Golden, E. Wasil, O. Balci, & W. Stewart (Eds.), *Impacts of recent computer advances on operations research* (pp. 407–421). New York: Elsevier.

Nance, R.E. & Overstreet, C.M. (1987). Diagnostic assistance using digraph representations of discrete event simulation model specifications. *Transactions of the SCS, 4* (1), 33–57.

Overstreet, C.M. & Nance, R.E. (1985). A specification language to assist in analysis of discrete event simulation models. *Communications of the ACM, 28* (2), 190–201.

Stucki, L.G. (1977). New directions in automated tools for improving software quality. In R. Yeh (Ed.), *Current trends in programming methodology,* Vol. 2 (pp. 80–111). Englewood Cliffs, NJ: Prentice-Hall.

Yeh, R.T. (1977). Verification of programs by predicate transformation. In *Current Trends in Programming Methodology,* Vol. 2 (pp. 228–247). Englewood Cliffs, NJ: Prentice-Hall.